Purpose

We built a 4-way street light system using a basys2 board and implementing a finite state machine using verilog to control it. Without any input, the lights should transition between each other for three seconds, one set of lights on and the other off in one state and the inverse of this for the other state. Once a button is pushed, this simulates a person wishing to use a crosswalk, the next state is then a crosswalk state where both sets of lights are off for five seconds allowing the person to cross the street. After that it then transitions back to the continuous state of switching back and forth for three seconds until the button is pushed again.

Some problems we faced were mostly from trying to implement the timer from the lab where we built the laser surgery module. We tried understanding how it worked and finally were able to make it work

Implementation

We began by designing our FSM to just cycle between light states for three seconds. Once that was achieved, we then began implementing an input represented as a button, if it was pressed or not. If it was pressed, it would then enter a state where the lights are off for 5 seconds and then returning to our previous cycle between the lights for three seconds.

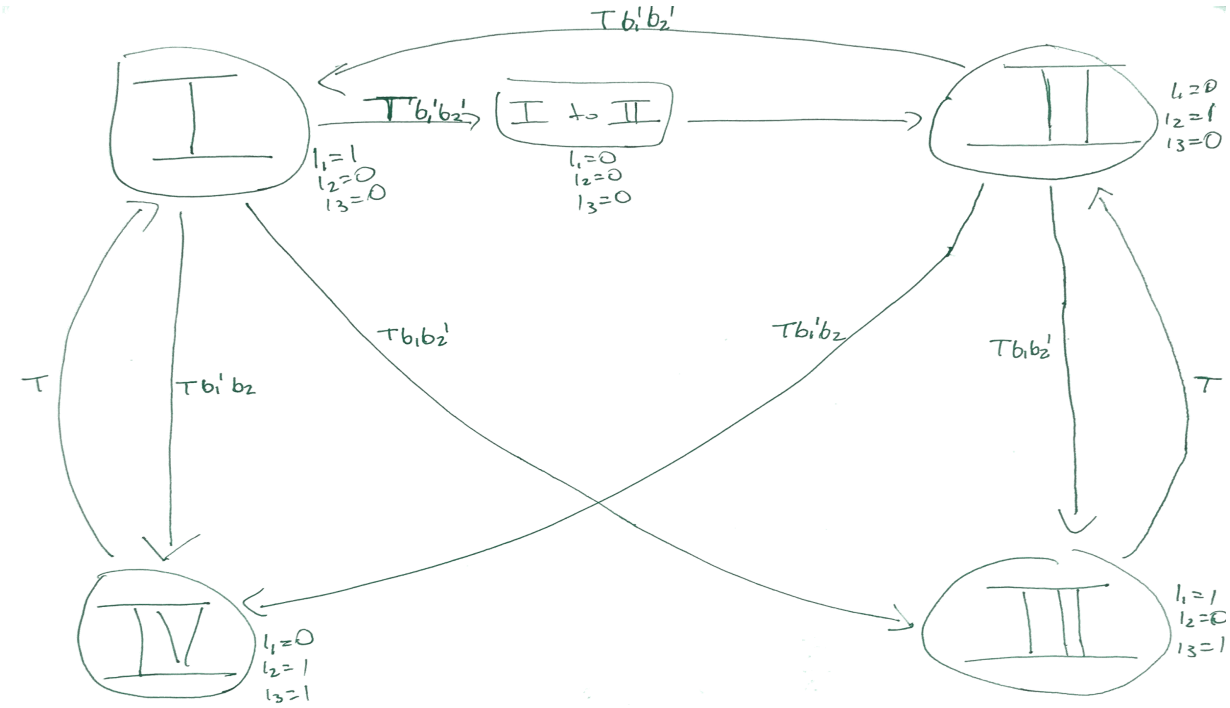
Once this was achieved we changed the third state to become two states because we introduced another button to indicate which crosswalk direction is being used, allowing us to choose which direction flow of traffic should go depending which crosswalk direction is being used.

System Design, Architecture, and Performance

Controller Design

We used two led lights to identify the direction of traffic flow, while another LED light is used to signal when the crosswalk is in use. We also utilized two buttons, and each button was designed to identify a certain crosswalk in a certain direction. Because we have the added button, that allows us to know which crosswalk is in use and allows certain traffic while pedestrians cross a certain crosswalk.

Circuit Schematic



State ONE

T	b1	b2	s1	s2	s3	l1	l2	l3	n1	n2	n3
0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	1
1	0	1	0	0	0	1	0	0	1	0	0
1	1	0	0	0	0	1	0	0	0	1	1
1	1	1	0	0	0	1	0	0	0	0	1

State ONEtoTWO

T	b1	b2	s1	s2	s3	l1	l2	l3	n1	n2	n3
0	0	0	0	0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0	0	0	1	0
0	1	0	0	0	1	0	0	0	0	1	0
0	1	1	0	0	1	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	0	0	0	1	0
1	1	0	0	0	1	0	0	0	0	1	0
1	1	1	0	0	1	0	0	0	0	1	0

State TWO

T	b1	b2	s1	s2	s3	l1	l2	l3	n1	n2	n3
0	0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	1	0	0	1	0	0	1	0
0	1	0	0	1	0	0	1	0	0	1	0
0	1	1	0	1	0	0	1	0	0	1	0
1	0	0	0	1	0	0	1	0	0	0	0
1	0	1	0	1	0	0	1	0	1	0	0
1	1	0	0	1	0	0	1	0	0	1	1
1	1	1	0	1	0	0	1	0	0	0	0

State THREE

T	b1	b2	s1	s2	s3	l1	l2	l3	n1	n2	n3
0	0	0	0	1	1	1	0	1	0	1	1
0	0	1	0	1	1	1	0	1	0	1	1
0	1	0	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	1	0	1	0	1	1
1	0	0	0	1	1	1	0	1	0	1	0
1	0	1	0	1	1	1	0	1	0	1	0
1	1	0	0	1	1	1	0	1	0	1	0
1	1	1	0	1	1	1	0	1	0	1	0

State FOUR

T	b1	b2	s1	s2	s3	l1	l2	l3	n1	n2	n3
0	0	0	1	0	0	0	1	1	1	0	0
0	0	1	1	0	0	0	1	1	1	0	0
0	1	0	1	0	0	0	1	1	1	0	0
0	1	1	1	0	0	0	1	1	1	0	0
1	0	0	1	0	0	0	1	1	0	0	0
1	0	1	1	0	0	0	1	1	0	0	0
1	1	0	1	0	0	0	1	1	0	0	0
1	1	1	1	0	0	0	1	1	0	0	0

$$b_1 = (S_1' S_2' S_3' + S_1' S_2 S_3)$$

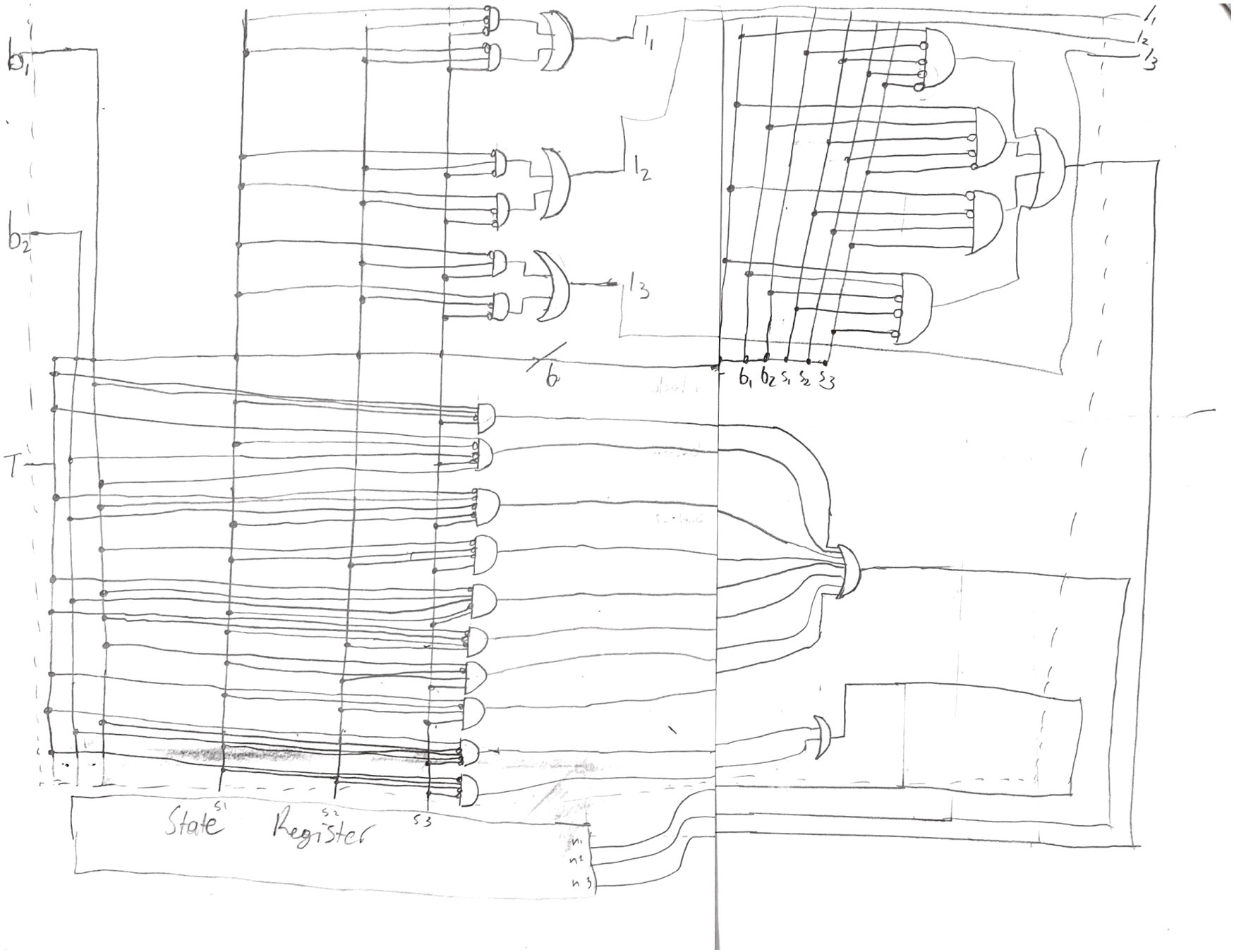
$$b_2 = (S_1' S_2 S_3' + S_1 S_2' S_3')$$

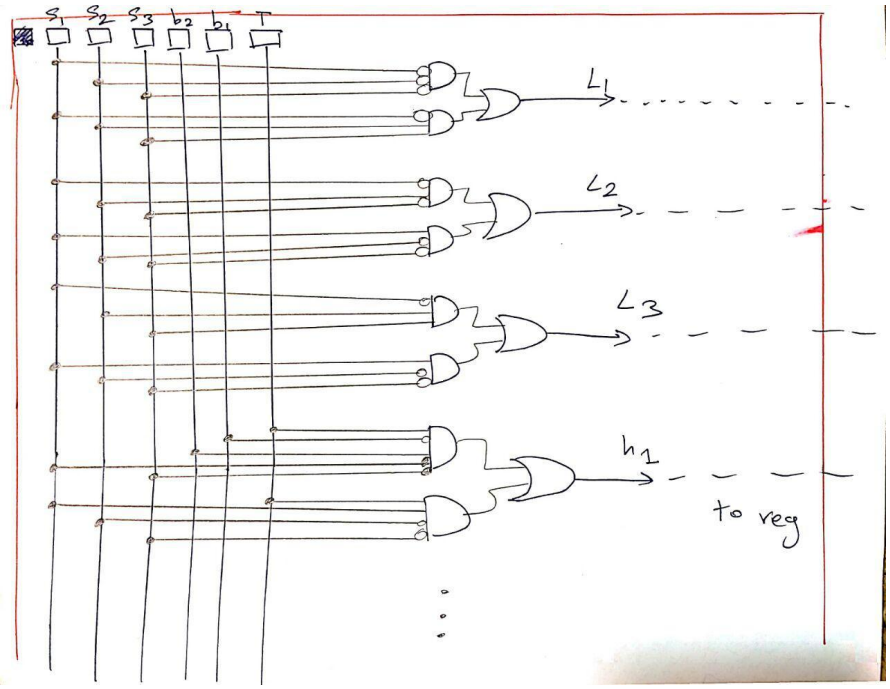
$$b_3 = (S_1' S_2 S_3 + S_1 S_2' S_3')$$

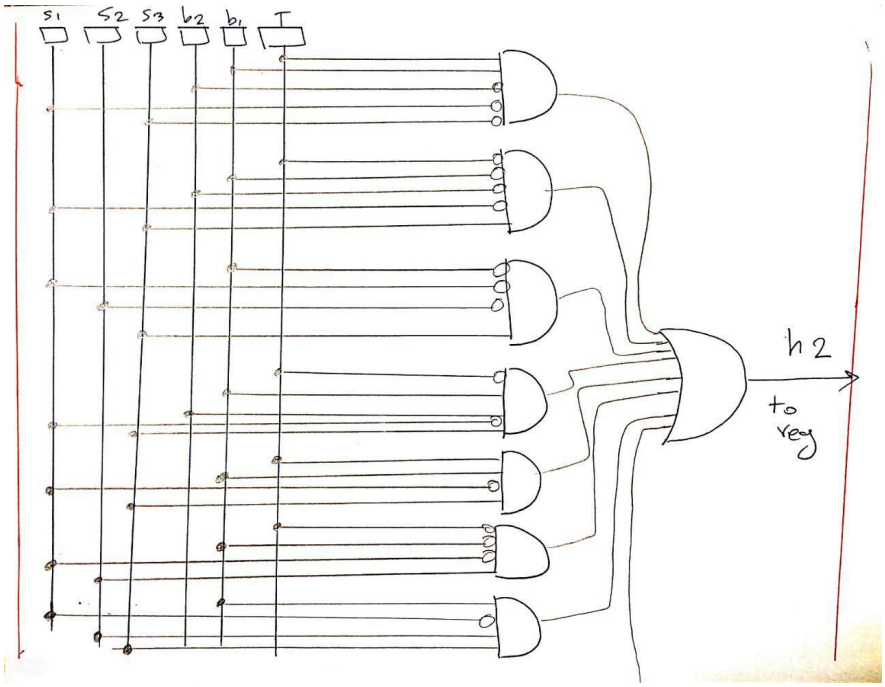
$$n_1 = (T b_1' b_2 S_1' S_3' + T' S_1 S_2' S_3')$$

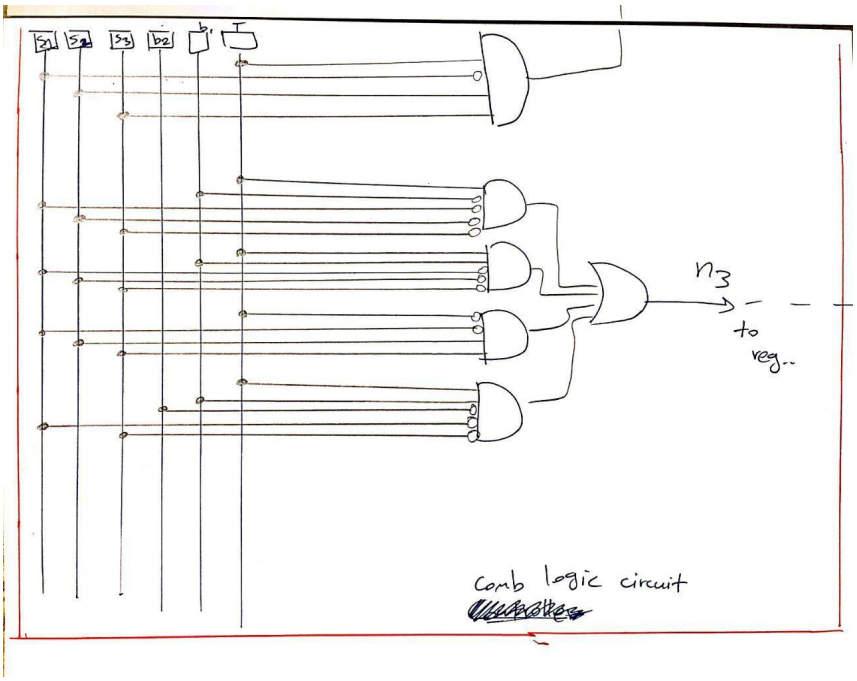
$$n_2 = (T b_1 b_2' S_1' S_3' + T' b_1' b_2' S_1' S_3 + b_1' S_1' S_2' S_3 + T' b_1 b_2 S_1' S_3 +$$
$$T b_1 S_1' S_3 + T' b_1' S_1' S_2 + b_1 S_1' S_2 S_3 + T S_1' S_2 S_3)$$

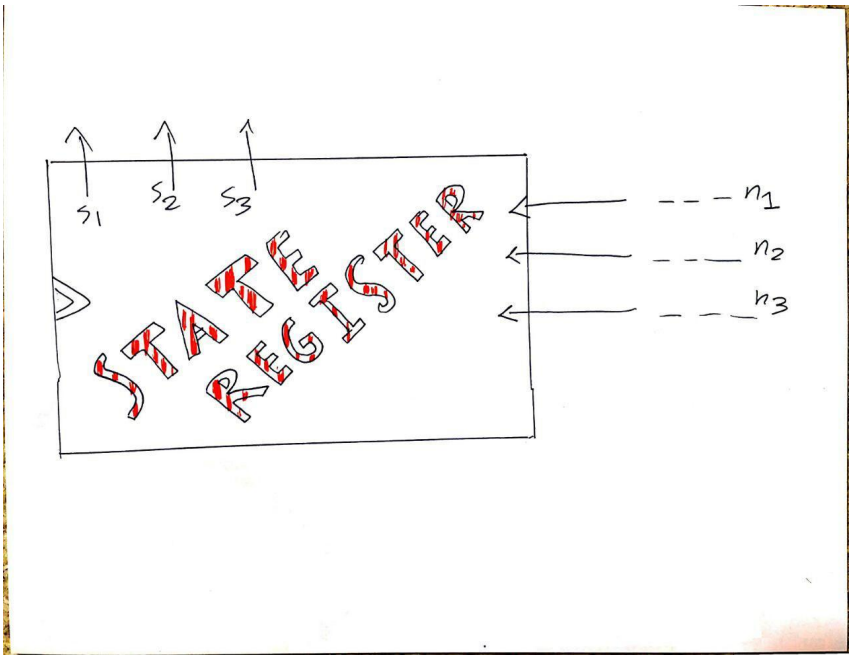
$$h_3 = (T b_2' S_1' S_2' S_3' + T b_1 S_1' S_2' S_3' + T' S_1' S_2 S_3 + T b_1 b_2' S_1' S_3')$$


b1
b2
T
l1
l2
l3
6
b1 b2 s1 s2 s3
State Register
s1
s2
s3
n1
n2
n3


s_1 s_2 s_3 b_2 b_1 T
L_1
L_2
L_3
h_1
to reg


s_1 s_2 s_3 b_2 b_1 T
h_2
to reg


s_1
s_2
s_3
b_2
b_1
n_3
to reg..
comb logic circuit


s_1
s_2
s_3
STATE REGISTER
n_1
n_2
n_3

Verilog Code

```
`timescale 1ns / 1ps

module street_lights #(
 parameter NBITS = 32
)
(
input wire b_one ,
input wire b_two ,
input wire clk ,
output reg light_one,
output reg light_two,
output reg light_three
);

wire timer;
wire b_pressed_one;
wire b_pressed_two;
reg select;
reg [2:0] current_state ;
reg [2:0] next_state ;
reg reset;
reg init;

wire [NBITS-1:0] cnt_ini ;
wire [NBITS-1:0] cnt_rst ;
wire [NBITS-1:0] cnt_rst_two;


// ---------------------------------------
// Sequential logic
// ---------------------------------------
always @(posedge clk) begin
current_state = next_state ;
end


// ---------------------------------------
// Comb. Logic
// ---------------------------------------
```

```
assign cnt_ini = 32'h0000 ;
assign cnt_rst = 32'h8F0D180; // 10 secs ( 25 MHZ internal clock )
assign cnt_rst_two = 32'hEE6B280;

// ---------------------------------------
// Comb. Logic - FSM
// ---------------------------------------
localparam ONE = 3'b000 ;
localparam ONEtoTWO = 3'b001 ;
localparam TWO = 3'b010 ;
localparam THREE = 3'b011;
localparam FOUR = 3'b100;
localparam transitionlol = 3'b101;

always @( current_state ) begin
case (current_state)
 ONE : begin
    light_one = 1'b1;
       light_two = 1'b0;
       light_three = 1'b0;
    reset = 1'b0;
       init = 1'b0;
       select = 1'b0;

    if (timer == 1) begin
            if(b_pressed_one == 1 && b_pressed_two == 0) begin
                  next_state = THREE;
            end
            else if(b_pressed_one == 0 && b_pressed_two == 1)begin
                  next_state = FOUR;
            end
            else begin
                  next_state = ONEtoTWO;
            end
       end
    else begin
        next_state = ONE;
    end
 end

 ONEtoTWO : begin
    light_one = 1'b0;
```

```
       light_two = 1'b0;
       light_three = 1'b0;
    reset = 1'b1;
       init = 1'b1;
       select = 1'b0;
    next_state = TWO;
end

 TWO: begin
    light_one = 1'b0;
       light_two = 1'b1;
       light_three = 1'b0;
    reset = 1'b0;
       init = 1'b0;
       select = 1'b0;

    if (timer == 1) begin
            if(b_pressed_one == 1 && b_pressed_two  == 0) begin
                  next_state = THREE;
            end
            else if(b_pressed_one == 0 && b_pressed_two == 1) begin
                  next_state = FOUR;
            end
            else begin
                  next_state = ONE;
            end
       end
    else begin
        next_state = TWO;
    end
 end

THREE : begin
      light_one = 1'b1;
      light_two = 1'b0;
      light_three = 1'b1;
      reset = 1'b0;
      init = 1'b1;
      select = 1'b1;

      if (timer == 1) begin
            next_state = TWO;
```

```
      end
      else begin
            next_state = THREE;
      end
end

FOUR : begin
      light_one = 1'b0;
      light_two = 1'b1;
      light_three = 1'b1;
      reset = 1'b0;
      init = 1'b1;
      select = 1'b1;

      if (timer == 1) begin
            next_state = ONE;
      end
      else begin
            next_state = FOUR;
      end
end

 default: begin
      light_one = 1'b0 ;
      light_two = 1'b0;
      light_three = 1'b0;
      reset = 1'b1;
      select = 1'b0;
      next_state = ONE ;
 end
endcase
end

// ---------------------------------------
// Timer instantiation
// ---------------------------------------
timer_st #( .NBITS(NBITS) ) timerst (
.timer(timer),
.clk(clk),
.reset(reset) ,
.select(select),
.cnt_ini(cnt_ini) ,
```

```
.cnt_rst(cnt_rst) ,
.cnt_rst_two(cnt_rst_two)
);

// button pressed instantiation
buttonPressed buttonPressed(
      .clk(clk),
      .b(b_one),
      .init(init),
      .b_pressed(b_pressed_one)
);

buttonPressed_two buttonPressed_two(
      .clk(clk),
      .b(b_two),
      .init(init),
      .b_pressed(b_pressed_two)
);

endmodule

module buttonPressed(
input wire clk,
input wire b,
input wire init,
output wire b_pressed
);

reg pressed;

always @(posedge clk) begin
      if (b == 1) begin
            pressed = 1'b1;
      end

      if (init == 1) begin
            pressed = 1'b0;
      end
end

assign  b_pressed = pressed;
```

```
endmodule


module buttonPressed_two(
input wire clk,
input wire b,
input wire init,
output wire b_pressed
);

reg pressed;

always @(posedge clk) begin
      if (b == 1) begin
            pressed = 1'b1;
      end

      if (init == 1) begin
            pressed = 1'b0;
      end
end

assign  b_pressed = pressed;

endmodule


module flopr #( parameter NBITS = 16 )(
 input clk,
input reset,
input [NBITS-1:0] cnt_ini,
input [NBITS-1:0] nextq,
output[NBITS-1:0] q
);

reg [NBITS-1:0] iq ;


always @(posedge clk) begin
 if (reset) begin
iq <= cnt_ini ;
 end
```

```
 else begin
iq <= nextq;
 end
end
assign q = iq ;
endmodule

module comparatorgen_st #( parameter NBITS = 16 )(
output wire r ,
input wire[NBITS-1:0] a ,
input wire[NBITS-1:0] b );
wire [NBITS-1:0] iresult ;
genvar k ;
generate
for (k=0; k < NBITS; k = k+1)
begin : blk
 xor c1 (iresult[k], a[k], b[k] ) ;
end
endgenerate
// Reduction plus negation
assign r = ~(|iresult);
endmodule

module fulladder_st(
output wire r,
output wire cout,
input wire a,
input wire b,
input wire cin
) ;
assign r = (a ^ b) ^ (cin) ;
assign cout = (a & b) | ( a & cin ) | ( b & cin ) ;
endmodule

module addergen_st #( parameter NBITS = 16 )(
output wire[NBITS-1:0] r ,
output wire cout ,
input wire[NBITS-1:0] a ,
input wire[NBITS-1:0] b ,
input wire cin ) ;
wire [NBITS:0] carry;
assign carry[0]= cin ;
```

```
genvar k ;
generate
for (k=0; k < NBITS; k = k+1)
begin : blk
 fulladder_st FA (
 .r(r[k]),
 .cout(carry[k+1]),
 .a(a[k]),
 .b(b[k]),
 .cin(carry[k]) ) ;
end
endgenerate
assign cout = carry[NBITS] ;
endmodule


module adder #( parameter NBITS = 16 )(
input [NBITS-1:0] q ,
input [NBITS-1:0] cnt_ini ,
input [NBITS-1:0] cnt_rst ,
output[NBITS-1:0] nextq,
output tick
);
wire same ;
wire[NBITS-1:0] inextq;
// -------------------------------------------------
// inextq = q + 1 ;
// -------------------------------------------------
addergen_st #(.NBITS(NBITS))
nextval ( .r(inextq), // Next value
 .cout(), // Carry out - Don't use
 .a(q), // Current value
 .b(16'b0000_0001), // Plus One
 .cin(16'b0000_0000) ) ; // No carry in
// -------------------------------------------------
// Are inextq and cnt_rst equal ?
// -------------------------------------------------
comparatorgen_st #(.NBITS(NBITS))
comparator (
.r(same) ,
.a(inextq),
.b(cnt_rst) );
```

```
// ------------------------------------------------------------------
// If they are the same produce a tick and set the value for nextq
// ------------------------------------------------------------------
assign tick = (same) ? 'd1 : 'd0 ;
assign nextq = (same) ? cnt_ini : inextq ;
endmodule

module timer_st #(
 parameter NBITS = 32
)
(
 output wire timer ,
 input wire clk ,
 input wire reset,
 input wire select,
 input [NBITS-1:0] cnt_ini ,
 input [NBITS-1:0] cnt_rst ,
 input [NBITS-1:0] cnt_rst_two
);
wire [NBITS-1:0] q ;
wire [NBITS-1:0] qnext ;
reg [NBITS-1:0] count ;

always @(posedge clk) begin
      if(select == 0)begin
            count = cnt_rst;
      end
      else if (select == 1) begin
            count = cnt_rst_two;
      end
end
// Compute the next value
adder #( .NBITS(NBITS ) )
 c1 (.q(q),
.cnt_ini(cnt_ini),
.cnt_rst(count),
.nextq(qnext),
.tick(timer) );
// Save the next state
flopr #( .NBITS(NBITS ) )
 c2 (.clk(clk),
.reset(reset),
```

```
.cnt_ini(cnt_ini),
.nextq(qnext),
.q(q) );
endmodule
```

YouTube Link

https://youtu.be/mn1i19AXdI4

Design Problems

One of the main problems we faced during implementation was with our original design. We originally decided to utilize the digital screen to show which direction traffic was going. We decided to scratch that idea because we couldn't get the clock to synchronize and find a way to select which segments to illuminate.

Another issue we came across was with the Finite State Machine. We realised when we were implementing the code with would always skip state 4 after state 3 and go to state 1. We realized that we didn't initialize it to the proper bit size, therefore the machine couldn't go to that extra state. We fixed it by changing the bit width of the registers.

Conclusion

The design we used helped us simplify the process at which the circuit operates. We utilized many skills that we attained in this course, and are very glad that we were able to get the final product to operate both successfully and efficiently. We had minor issues as discussed above, but we were able to overcome them.

In conclusion, we created a working street light system in verilog while using a FPGA board that has a cross walk feature that allows certain traffic to pass as a certain crosswalk direction is in use.

One way of optimizing our Finite State Machine is to have fewer states. Another is to allow our system to let users use both crosswalk directions at the same time. In addition, we can implement more features to allow easy accessibility for users, e.g, a timer so that users know how long they have to cross the street..